Honours Project (CCIS)

INTERIM REPORT

Submitted for the Degree of:
BSc Computer Games (Software Development)

Project Title: The performance implications of adapting a physics engine to function deterministically across multiple platforms.

Name:

Programme: Computer Games (Software Development)

Matriculation Number:

Project Supervisor:

Second Marker:

Word Count: 4342

1. Contents

2. Introduction

2.1. Computers are Deterministic

While computers are technically incapable of random behaviour, as each individual instruction they execute is deterministic (Weaver *et al.*, 2013), a program, viewed in isolation, can exhibit non-deterministic behaviour. This is because background processes, memory timings, and memory allocation (Kaeli and Sachs, 2009) (Henning, 2000) (Devietti *et al.*, 2009) can all influence the precise behaviour of a single program making it appear non-deterministic, when in reality, the system is simply complex enough to appear random, if not fully understood.

As an individual program will often have to share resources with other processes running on the same machine, it should be noted that this can result in variations in timings and the precise order of instructions a CPU executes (Weaver *et al.*, 2013), however it will not affect the program's output. Therefore, such a program will still be considered *deterministic* throughout this report.

2.2. Cross-Platform Determinism

Although possible to create a program that runs deterministically on a given computer, it is a complex task to run that program deterministically across multiple platforms. This is due to the inaccuracies in how these platforms perform floating-point arithmetic (Whitehead and FitFlorea, 2011) (Hillesland and Lastra, 2004) (Collavizza *et al.*, 2016), as well as the inconsistencies introduced between different compilers and optimisation levels, while compiling the same source code (Monniaux 2008) (Hoste *et al.*, 2008) (Kreinin, 2008) (Dawson, 2013) (Sawaya *et al.*, 2017).

This source of non-determinism has been the cause of serious issues, from security threats (Andrysco et al. 2015) and incorrect stock information in Vancouver, to the failure of a Patriot Missile during the Gulf War (Kneusel, 2017).

While the results of non-determinism in floating-point arithmetic are not always catastrophic, they do often cause problems in a variety of industries.

2.3. Importance of Physics Engines

Physics engines play an essential role in many fields, including robotics (Rönnau *et al.*, 2013) (Ivaldi *et al.*, 2014) (León *et al.*, 2010), simulations, and computer games.

Despite the lack of determinism in these physics engines, developers still need to create functionality that behaves predictability. This is often a problem when simulations require repeatability.

Rohde (2009) discusses the *US Army Engineer Research and Development Center* developing unmanned ground vehicles. During their development, simulations captured video for playback as this was the only reliable way to replay their simulation. During Peitso and Brutzman's (2018) work on augmented reality for decision-making and safety, they found that the repeatability of their simulations was unreliable. Both of these research groups could have benefited from a deterministic physics engine.

2.4. Physics Engines in Computer Games

The benefits of a deterministic physics engine could also be applied in computer games. Unity and Unreal Engine 4 are two of the most used game engines (Christopoulou and Xinogalos, 2017) (Kushnir, Koman, and Shuvar, 2018). Both of these engines have implemented Nvidia's PhysX SDK for their 3D physics simulations (Juliani *et al.*, 2018) (Epic Games, 2019), which is non-deterministic (Martínez-Franco *et al.*, 2018).
As such, developers cannot rely on replays being consistent across different platforms.

Currently, if a simulation is to be replayed perfectly, there are only two ways to guarantee this. One is to store the objects at every frame, which results in large replay files. The other is to save the initial frame and then only the changes of subsequent frames, this reduces file size but makes scrubbing tedious as each frame is dependant on the previous, all the way back to the initial frame.

Lobb (*et al.*, 2004) of Nintendo of America Inc. filed a patent outlying their design for a game replay system which saves periodic keyframes in their entirety and, for the frames in between, only records player inputs. As this method contains key frames it would allow for smooth scrubbing. It is also a compromise between accuracy and file size, only recording the positions of objects at each key frame and allowing the physics engine to recalculate the rest. Although any inaccuracies caused by this method will be negligible to most games, Kelly (*et al.*, 2018) found that esports titles use replays to encourage crowd hype, and as such, will likely prefer the precision currently only achievable by storing every frame.

2.5. Fixed-Point Arithmetic

Box2D and Newton Dynamics are two physics engines which were developed with the intention of being deterministic (Catto, 2019a) (Newton Dynamics, 2019). However, neither can be considered deterministic across multiple platforms as they both make extensive use of floating-point arithmetic.

As discussed, floating-point arithmetic is not deterministic across platforms and compilers.
However, due to the lack of rounding involved, integers are deterministic across platforms (Ducas *et al.*, 2019).
As Andrysco (*et al.*, 2015) said, it is not feasible to limit programmers to only using integers. The answer to this is fixed-point. A fixed-point number is stored as an integer but interpreted as a real number through the use of a library. The result of which is a representation of a real number that is deterministic across platforms.

2.6. Performance

Due to the additional steps required to compute the result of a fixed-point operation (Sanisalo and Kero, 2019), it is speculated that adapting a physics engine to use a fix-point library will result in a performance deficit.

2.7. Question

This leads into the goal of the project, answering the question:

"*What are the performance implications of adapting a physics engine to function deterministically across multiple platforms?*"

3.Literature & Technology Review

3.1. Literature Review

3.1.1. Non-determinism in Physics Engines

As mentioned in the introduction, both Unity and Unreal Engine 4 use Nvidia PhysX for their 3D simulations. This physics engine, along with other popular options such as ODE and Bullet, are all non-deterministic in nature (Hämäläinen *et al.*, 2017) (Martínez-Franco *et al.*, 2018).

Physics engines can become non-deterministic through several means. The use of pseudorandom number generators (PRNGs) in the solvers of these engines is fairly common, for example, Bullet details how the PRNG is used in their pipeline (Coumans, 2019). Solvers are responsible for resolving the physics collision after they have been detected. It is clear how the use of a PRNG during this step can lead to non-deterministic outcomes.
However, since PRNGs are based on a seed (Michaelis, Meyer, and Schwenk, 2013) and, given the same seed, all random numbers generated are entirely predictable. An engine need only expose to its users a way of setting this seed to be capable of achieving repeatable results.

Additionally, a physics engine must implement a fixed time step (Vasudeva and Bhalla, 2004), this is in contrast to a variable time step that animations and rendering are based on. With variable, the time taken to perform the previous cycle influences the next, whereas fixed time step always uses the same value for time passed and it is the number of cycles which varies instead. This approach results in more stable physics simulations (Erleben, 2004).

The next obstacle a physics engine must overcome to be deterministic is also related to the solver, specifically, the information which is passed into it. The order in which bodies are processed can affect the outcome of the physics simulation (Karagoz, 2000) (Nieter and Cary, 2004).
If a simulation is to be deterministic, this order must be controlled. For example, in the C++ Standard Library, an unordered set returns elements in an unspecified order (Cpp Reference, 2019a), and Java HashMaps order elements based on their hashcode which is, in turn, based on their address in memory (Okano *et al.*, 2019). Since the memory address an object is assigned cannot be guaranteed (Devietti *et al.*, 2009), its order in such a list also cannot be guaranteed. Sets such as these should not be used to reference the bodies a physics solver will iterate over.

Finally, there are very few physics engines which have managed to solve all of the issues above. One example is Newton Dynamics. Newton "implements a deterministic solver, which is not based on traditional LCP or iterative methods" (Newton Dynamics, 2019). The reliability and accuracy of Newton's deterministic solver is verified by the work of Hummel (*et al.*, 2012).

Another example is Box2D, used by Unity for its 2D physics simulations (Pereira, 2014). According to Box2D's author Erin Catto (2019a) and Unity's Developer Relations Lead Ricardo Arango (2018), this engine is deterministic for the same binary.

Analysis of both Newton Dynamics' (Jerez, 2019) and Box2D's (Catto, 2019b) source code reveals the extensive use of floating-point numbers. This is not unexpected given the performance and flexibility benefits of floats (Williams *et al.*, 2009), but it does mean that despite Newton and Box2D both being deterministic on a given machine, their determinism still cannot be guaranteed across multiple platforms due to the non-deterministic nature of floats.

3.1.2. Issues with Floating-point Arithmetic

Floating-point numbers, regardless of their precision, are represented with a finite number of bits. This means that they are, of course, an approximation of the real numbers they represent (Whitehead and Fit-Florea, 2011). Real numbers are rounded before being represented as a float. The IEEE established a standard for floating-point arithmetic called the IEEE 754 (Kahan, 1996). This standard details which operators hardware must support, the minimum precision of the resulting float after these operations are performed, as well as how much it may deviate from its real number equivalent (Hillesland and Lastra, 2004).
Despite these guidelines, the limited precision of floats means that the results of their operations can be "significantly different" when compared to the same operations performed on real numbers (Collavizza *et al.*, 2016).
This limitation of floating-point precision does not prevent a program achieving cross-platform determinism. That problem results from different architectures achieving inconsistent results while approximating these same real number equivalent operations.

Monniaux (2008) found that floating-point computations can vary subtly across common hardware platforms and that the only way to guarantee their consistency is to use the exact same machine code across platforms. With that said, Hoste (*et al.*, 2008) was more specific in his findings, stating that "modern compilers implement a large number of optimizations which all interact in complex ways" which can result in subtle differences in floating-point arithmetic between each optimisation level. However, Kreinin (2008) and Dawson (2013) show that even with the lowest levels of optimisation, different compilers can still produce differing results.

This is backed up by Sawaya's (*et al.*, 2017) research who found that using the same source code to produce the required binaries for each target platform while maintaining determinism is technically possible, but highly impractical. Sawaya and their collaborators found that different compiler optimisations will result in different floating-point results, but by "severely limiting" the use of compiler optimisations they were able to achieve consistent floating-point arithmetic results across multiple platforms. However, this came at a performance cost of approximately 3-5 times, a sacrifice that many programmers will be unable to make. It should be noted that such a sacrifice was not always necessary, as sometimes the optimization produced identical results, but this behaviour is difficult to predict. (See appendix A.)

These are some obvious limitations for any program which requires determinism and utilizes floating-point operations. The program may only run the same machine code on it's target platforms, limiting the number of platforms it can reach, or suffer a severe performance cost.

While all the resulting descrepenses discussed above may be very small and within a range that the IEEE deem acceptable, it is important to note that these small differences can amplify over time and result in simulations quickly becoming out-of-sync (Pool, 1989).

3.1.3. Fixed-point Arithmetic

To overcome the inconsistencies associated with floating-point arithmetic, one could simply avoid them and rely solely on integers in their calculations. Integer arithmetic is indeed deterministic across compilers and platforms, regardless of optimisation levels (Ducas *et al.*, 2019). However, as Andrysco (*et al.*, 2015) said, it is not feasible to limit programmers to only using integers, certain calculations absolutely require a representation of the numbers between integers.

The solution to this problem of non-determinism in floating-point arithmetic is indeed, to use integers, but to have them represent real numbers. The traditional representation of a floatingpoint number can be seen in Figure 1, here, the significant digits (significand) are represented by a portion of the available binary digits. These significant digits are then multiplied to represent much larger or smaller number, in a way akin to scientific notation.

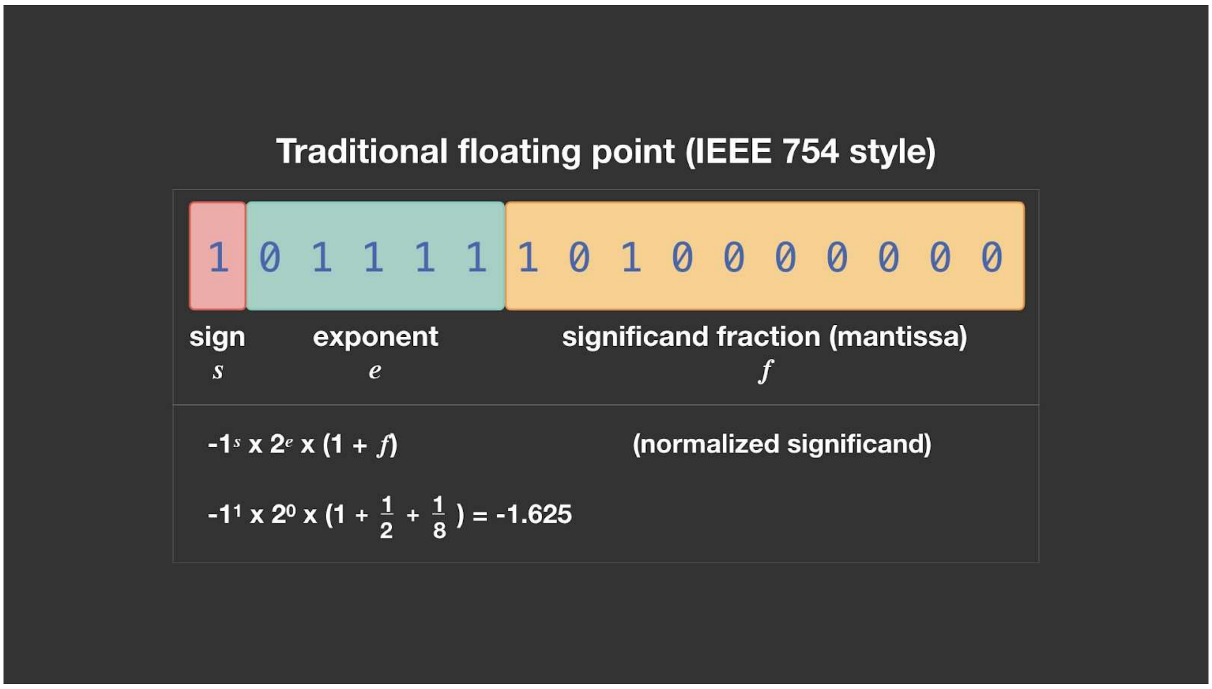

Traditional floating point (IEEE 754 style)
1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0
sign
s
exponent
e
significand fraction (mantissa)
f
$-1^s \times 2^e \times (1 + f)$
(normalized significand)
$-1^1 \times 2^0 \times (1 + \frac{1}{2} + \frac{1}{8}) = -1.625$

Figure 1: Floating-point binary representation.
(Johnson, J., 2018)

In contrast, a fixed-point number is still interpreted as an integer to the compiler, but, through code, is handled differently (see Figure 2). A fixed-point number has its binary digits split (not always in half) with one portion representing the decimal digits before the decimal point, and the other portion after. With a specifically written library, an integer can be treated as a real number without any of the non-deterministic issues of a floating-point.

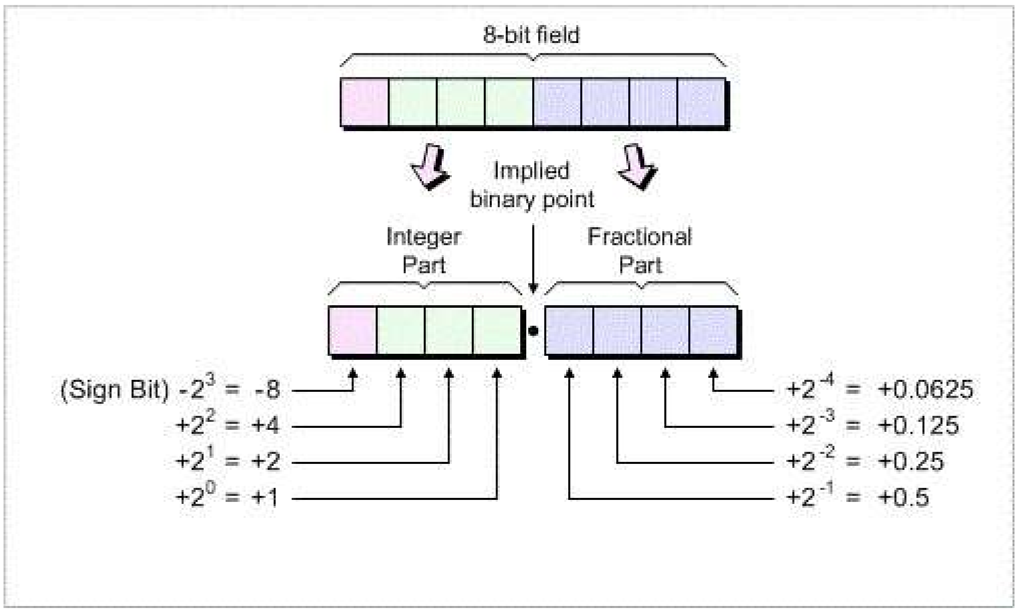


Figure 2: Fixed-point binary representation.
(Arar, 2017)

3.2. Technology Review

3.2.1. Physics Engine

The project requires the use of a freely available, open-source physics engine. Creating a custom physics engine would have granted more control over the arithmetic used, ensuring its determinism, but the real-world performance metrics that could only be obtained from using a well-established engine was deemed to be critical to the study. The drawbacks of obtaining this real-world data is that care must be taken to ensure that no floating-point arithmetic remains in the engine.
The engine chosen must be open-source so that the floating-point arithmetic it uses can be replaced with a deterministic fixed-point library.
It should already be deterministic on the same platform with the same binary, as this research will focus on the cross platform deterministic limitations caused by floats, not the many other reasons a physics engine can be non-deterministic, as previously discussed.

There are two popular physics engines which meet the above criteria. Box2D (Catto, 2019b) and Newton Dynamics (Jerez, 2019). As discussed, Newton does not use PRNGs and has a deterministic solver (Newton Dynamics, 2019) and Box2D's determinism is verified by both its author and by Unity's Developer Relations Lead (Catto, 2019a) (Arango, 2018).
Either of these engines could have been chosen, but the smaller codebase of Box2D is a better fit for the scope of the project.

3.2.2. Fixed-point Library

An open-source fixed-point library will be used to replace the floating-point arithmetic in the physics engine. The library must be open-source so that it can be compiled alongside the physics engine and, as with the physics engine, creating such a library was outwith the scope of the project.

There are certain requirements which a fixed-point library must fulfill to be viable for the project. The library must be open-source, have sufficient arithmetic operators to execute all of the engine's physics calculations, and be implemented into an existing physics engine without requiring a dramatic overhaul of the engine's code.

Some preliminary experiments were conducted on the library's assessed below to determine their functionality as this information was not always available from their documentation.

There are many fixed-point libraries freely available on GitHub, three of which were evaluated before one was found which fulfilled all the above requirements.

The Compositional Numeric Library (CNL) (McFarlane, 2019) is a comprehensive fixed precision numeric library providing access to many data types not commonly included with programming languages, one of which is fixed-point. Upon initial analysis of CNL's source code, it became apparent that much of the functionality this library provides was not required by the project and the coding conventions, such as template meta-programming, made the library very obtuse (Veldhuizen, 2000). This could impede its implementation into a physics engine and, for that reason, was rejected for use in the project. It should be noted that a senior software developer may prefer this more sophisticated library and should be considered for future research.

Another library evaluated was FixedPoint (Trenkwalder, 2019). The scope of this library was far narrower than the previous, resulting in a simpler implementation style, but was missing a few arithmetic functions that would likely be required by a physics engine, namely, absolute value, linear interpolation, and some trigonometric functions. The library also lacked functions to easily convert to and from floating-points, a feature which would be convenient during development.

The final library considered, and ultimately settled on, was the FixPointCS library (Sanisalo and Kero, 2019), a library which was explicitly verified to be deterministic by the author. The library's source-code had a simpler style, was well documented, and contained a full suite of arithmetic functions which will likely cover everything required by a physics engine. It also implemented convenient functions for converting to and from floating-point, this would be essential for passing information onto the OpenGL pipeline to be rendered and would help with development and debugging.
The library also supported several additional features that were not a requirement for this project but could prove useful for future research. It has been written in three programming languages (C#, Java, and C++), supports both 64 and 32 bit instruction sets, and implements three variants of some of its more demanding operators (fast, faster, and precise).

3.2.3. IDE

The Microsoft Visual Studio 2019 IDE (Microsoft, 2019a) was chosen to develop the software. As shown from the literature review, different compilers and compiler optimisations can result in non-deterministic binaries (Hoste *et al.*, 2008) (Kreinin, 2008) (Dawson, 2013). Visual Studio supports the use of the Microsoft Visual C++, Clang C++, and GCC C++ compilers (Microsoft, 2019b). Microsoft also provides documentation on the optimization flags for the The Microsoft Visual C++ Compiler (Microsoft, 2017).

The software will be written in C++ as this language is optimised for speed (PEREZ, 2017) and as such, is a popular choice when the performance of an application is important. Additionally, all of the required libraries are available (Catto, 2019b) (Sanisalo and Kero, 2019) in this heavily documented language (CPlusPlus, 2019) (Cpp Reference, 2019b) (DevDocs, 2019), aiding in development.

4. Methods

The *software* being developed during this project consists of four binary files. These binaries will be discussed in detail later but it is important to note that they are not the contribution of this research. This software will only be used to gather the required data to answer the research question.

4.1. Development Model

Several software development models were researched and their effectiveness at guiding this project evaluated.

The first model evaluated was agile. In Beck's (*et al.*, 2001) twelve principles behind the agile manifesto, the highest priority is to satisfy the customer and to welcome changing requirements. As this project has a fixed goal and no stakeholders, this method was rejected.

The widely used waterfall model was accessed next (Petersen, Wohlin, and Baca, 2009). According to the work of Balaji, waterfall is recommended for larger development teams and can produce extensive, seldom used, documentation (Balaji, and Obaidy, 2016) (Balaji, and Murugaiyan, 2012). As this software is not the focus of the research and will be created by a single programmer, less emphasis will be put into the software development documentation. Given this, waterfall was also rejected.

Test-driven was the final model considered for use in this project. This model can be categorized by its short development cycle and with each requirement being assessed by a test case which is often written even before the implementation (Astels, 2003) (Beck, 2003). Maximilien and Williams (2003) found that a test-driven model can reduce the defect rate of software by 50%, compared to other models. The attention to testing and error-detection encouraged by the test-drive model will support this project as the correct functionality of this software hinges on very subtle differences in CPU arithmetic. The test-driven model also encourages regression testing, further increasing the reliability of the software it produces. Therefore, this model was chosen to guide development during this project.

4.2. Envisioned Functionality

The software created will consist of four binary files, each binary will run a physics simulation based on the exact same initial conditions.

There are subtle but important differences between these four binaries:
Two of the binaries will be created using conventional floating-point arithmetic and each will be compiled on a different compiler or with different compiler optimization settings in order to make them non-deterministic (Sawaya *et al.*, 2017).

The other two binaries will be compiled with the same compilers and optimization settings as the previous two, and will be created using a fixed-point arithmetic library. These binaries will be used to show that determinism can be achieved in a scenario that is impossible with floatingpoint.

Each of the two binaries that were compiled using the most optimised settings will be used to compare the performance of each simulation. This is to highlight the performance discrepancy between using floating-point primitives and a fixed-point arithmetic library, given that each binary is otherwise identical and each is able to perform optimally.

Since it is possible for the optimised floating-point and fixed-point binaries to simulate slightly different simulations, several environments will be created to minimise the effect that one simulation could chaotically lead to a more demanding simulation.

4.3. Development Environment

As discussed in the technology review, the Microsoft Visual Studio 2019 IDE (Microsoft, 2019a) will be used to develop the software as it supports three different compilers as well as compiler optimisation flags (Microsoft, 2019b) (Microsoft, 2017) which will allow the creation of non-deterministic binary files.

The Box2D physics engine, due to its relatively small code base and being deterministic for a single binary (Catto, 2019a; 2019b), will be used to run physics simulation with both floating and fixed-point arithmetic.
The fixed-point arithmetic will be implemented into Box2D using Sanisalo and Kero's (2019) FixPointCS.

Two additional libraries, while not strictly required, will also be used; the OpenGL Extension Wrangler, GLEW (Ikits *et al.*, 2019) and the OpenGL Mathematics Library, GLM (G-Truc Creation, 2019). These libraries will allow the physics objects being simulated by Box2D to be rendered, aiding in the development and debugging of the software. It is important to note that the visual aspect of this software is not a part of the research being conducted. Since a CPU and GPU overhead will be introduced by this, care will be taken to minimise the impact rendering has on CPU performance and to avoid a GPU bottleneck (Madougou *et al.*, 2016). This ensures that only differences in the physics engine result in overall differences in the software's performance. OpenGL instance rendering and simple shaders will be used to achieve this low overhead (Learn OpenGL, 2019). The system monitoring software HWiNFO will be used to monitor GPU performance during runs (HWiNFO, 2019).

4.4. Data Capture

The data captured during this experiment will be the times taken for the software to render each frame of the simulation. The high resolution clock, available from the C++ standard library (Cpp Reference, 2019c), will be used to accurately record system time before and after each software cycle. The difference between these values is known as a frame time and those frame times will be used in the formula below (Figure 3) to calculate the average frames per second during the simulation. This will be the performance metric of this experiment.

$$\textit{Average frames per second is given by } \frac{1}{\bar{x}},$$

$$\textit{where } \bar{x} = \frac{1}{n}\sum_{i=1}^{n} xi \textit{, and } xi \textit{ is the set of all frame times.}$$

Figure 3: Average frames per second formula.
An adaptation of the average formula (Wolfram Research, 2019).

4.5. Hypothesis

Based on the research found in the literature review, that floating-point arithmetic is nondeterministic across different platforms and for different binaries (Monniaux, 2008), unless compiler performance optimisations are limited (Sawaya *et al.*, 2017). It follows to ascertain the performance impact of a physics engine which achieved determinism through the use of a fixed-point library. It is hypothesised that:

"*The use of a fixed-point library will negatively affect the performance of a physics engine.*"

4.6. Statistical Analysis

As formally defined in Data Capture, the performance metric gathered by the software is the average number of frames per second the software is able to render during the simulation.

In order to determine which analysis technique should be applied to this data, a statistical decision tree was used and the following questions were answered:

1. *"How many outcome variables are you interested in for your study?"*
   One - Only the average frames per second of the binaries will be assessed.

2. *"Does the outcome of interest contain continuous measurements or categorical values?"*
   Continuous - Frames per second is a continuous measurement.

3. *"How many predictor variables do you have?"*
   One - The binaries only differ in their use of arithmetic.

4. *"Are the predictor variables continuous measurements, categorical values or both?"*
   Categorical - The type of arithmetic used is categorical.

5. *"For your categorical variables, how many possible values are there?"* Two - Those categories are either floating-point or fixed-point.

6. "Will you be collecting data from the same or different (sample/patient/individual) multiple times? Or will you be doing a mixture of same and different?"
   Same - The binaries will be identical apart from the arithmetic used.

This revealed that a dependant two-tailed T-test should be applied to analyse the data.

4.7. Sample Size

Due to variations in CPU performance between runs, caused by interrupt instructions (Weaver *et al.*, 2013), background processes, memory timings, and memory allocation (Kaeli and Sachs, 2009) (Henning, 2000) (Devietti *et al.*, 2009), sample size analysis was used to determine how many samples will be required to achieve a power of 0.8, effect size of 0.5, and a significance level of 0.05. A sample type was set to two sample (as the average frames per second of each binary will be compared) and alternative was set to two sided (as performance differences could oppose the hypothesis), which revealed that a sample size of 63.770, which will be rounded up to 64, will be required to achieve the power value of 0.8.

Therefore, the performance of both the floating-point and fixed-point binary will be captured 64 times each in order to lessen the impact of these variations in CPU performance.

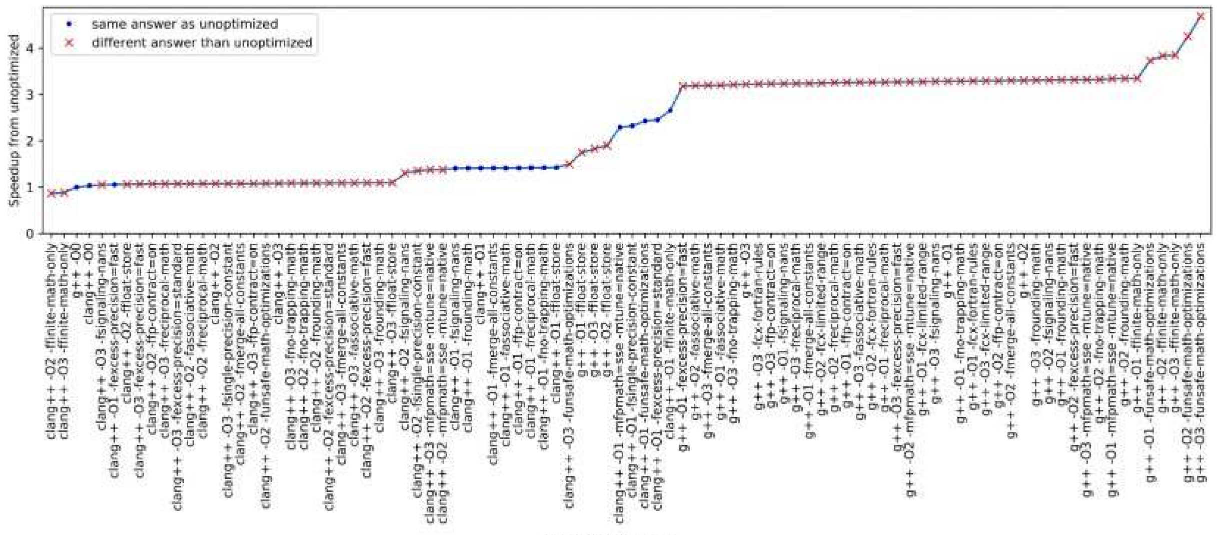
4.8. Ethical Considerations

The project will perform statistical analysis on quantitative data generated by the software. As there are no participants, it follows that there are no ethical issues to consider.

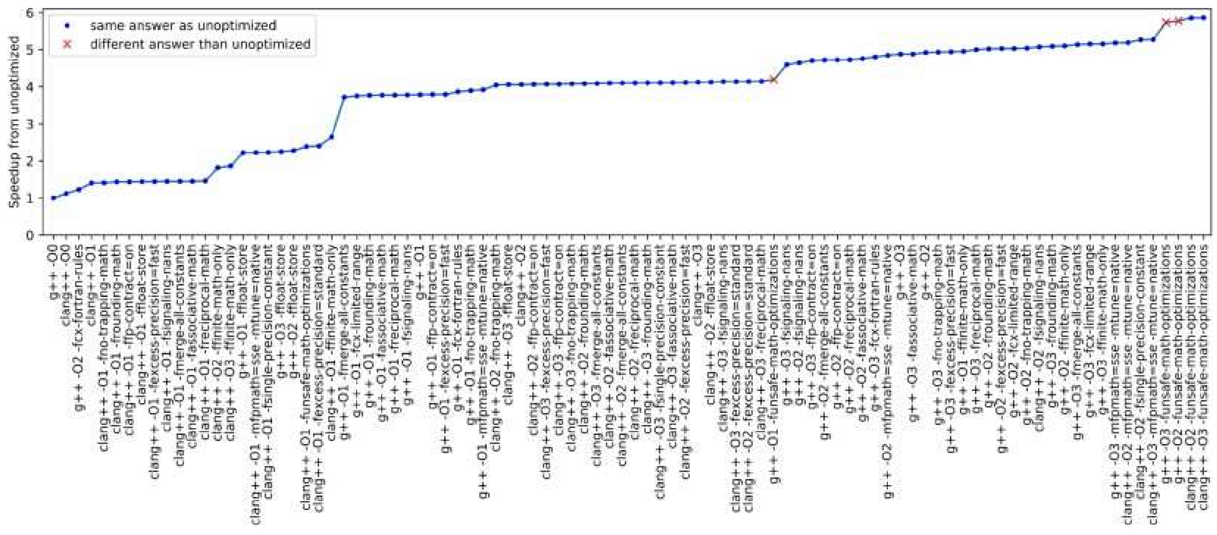
4.9. Gantt Chart

(See appendix B.)

5. Appendices



(a) TrianglePSylv

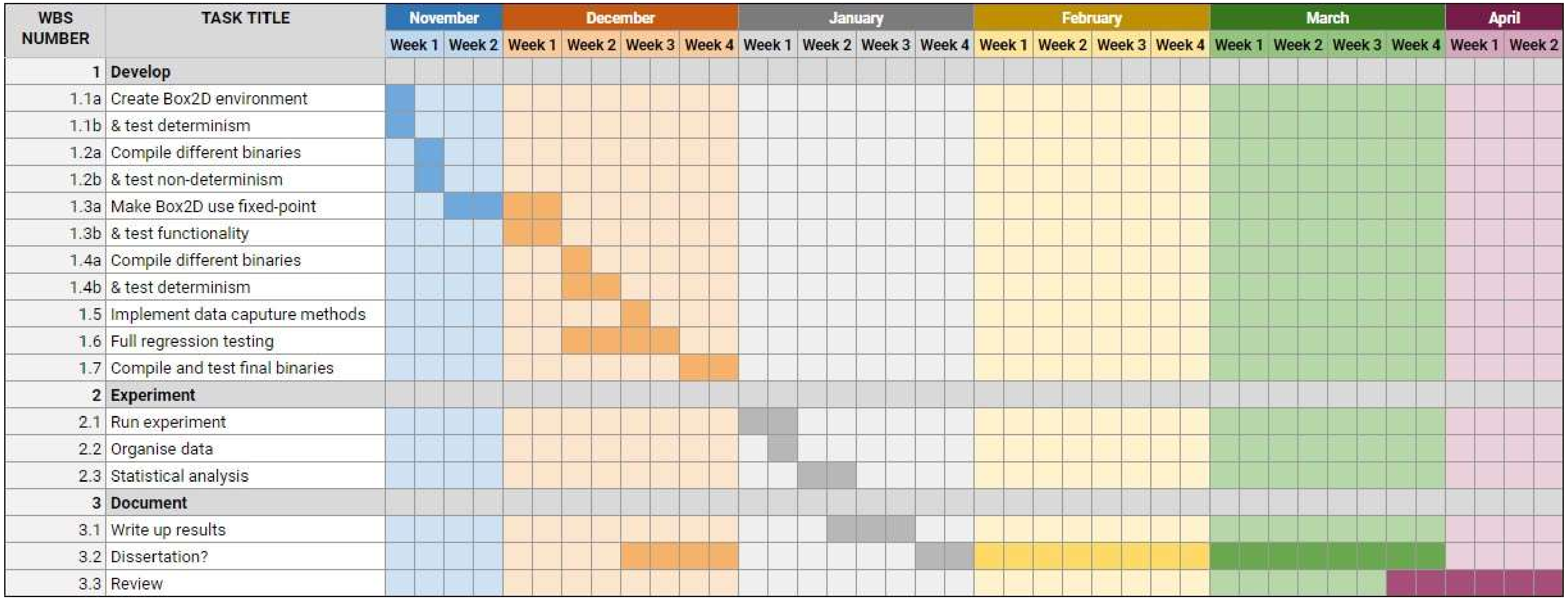


(b) TrianglePHeron

Appendix A: Compiler Optimisation Inconsistencies
(Sawaya *et al.*, 2017)

This chart shows two experiments on compiler floating-point accuracy. In each experiment, the results produced with compiler optimisations enabled were compared to compiler optimisations disabled. If the results were the same, it was marked with a blue cross; and if the results were different, it was marked with a red cross.

As it can be seen, compiler optimisations can have a significant impact on the accuracy of floating-point operations, but that the specific workload is also a very important factor. Leading to further evidence that it is hard to predict floating-point behaviour.



WBS NUMBER	TASK TITLE	November		December				January				February				March				April	
		Week 1	Week 2	Week 1	Week 2	Week 3	Week 4	Week 1	Week 2	Week 3	Week 4	Week 1	Week 2	Week 3	Week 4	Week 1	Week 2	Week 3	Week 4	Week 1	Week 2
1	**Develop**																				
1.1a	Create Box2D environment	■□																			
1.1b	& test determinism	■□																			
1.2a	Compile different binaries	□■																			
1.2b	& test non-determinism	□■																			
1.3a	Make Box2D use fixed-point		■■	■■																	
1.3b	& test functionality			■■																	
1.4a	Compile different binaries				■□																
1.4b	& test determinism				■■																
1.5	Implement data caputure methods					■□															
1.6	Full regression testing				■■	■■															
1.7	Compile and test final binaries						■■														
2	**Experiment**																				
2.1	Run experiment							■■													
2.2	Organise data							□■													
2.3	Statistical analysis								■■												
3	**Document**																				
3.1	Write up results								□■	■■											
3.2	Dissertation?					■■	■■				■■	■■	■■	■■	■■	■■	■■	■■	■■		
3.3	Review																		■■	■■	■■

Appendix B: Dissertation Gantt Chart

6. References

Andrysco, M., Kohlbrenner, D., Mowery, K., Jhala, R., Lerner, S. and Shacham, H., 2015, May. On subnormal floating point and abnormal timing. In 2015 IEEE Symposium on Security and Privacy (pp. 623-639). IEEE.

Arango, R., 2018. Determinism with 2D Physics - Unity. Unity Technologies. [viewed 13 November 2019]. Available from:
https://support.unity3d.com/hc/en-us/articles/360015178512-Determinism-with-2D-Physics

Arar, S., 2017. Fixed-Point Representation: The Q Format and Addition Examples - Technical Articles [online]. All About Circuits. [viewed 13 November 2019]. Available from: https://www.allaboutcircuits.com/technical-articles/fixed-point-representation-the-q-formatand-addition-examples/

Astels, D., 2003. Test driven development: A practical guide. Prentice Hall Professional Technical Reference.

Balaji, S. and Murugaiyan, M.S., 2012. Waterfall vs. V-Model vs. Agile: A comparative study on SDLC. International Journal of Information Technology and Business Management, 2(1), pp.26-30.

Balaji, S. and Obaidy, M.A., 2016, March. Project characteristics used for methodology selection to develop the software project. In 2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT) (pp. 3570-3573). IEEE.

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R. and Kern, J., 2001. Manifesto for agile software development.

Beck, K., 2003. Test-driven development: by example. Addison-Wesley Professional.

Catto, E., 2019a. FAQ · erincatto/Box2D Wiki [online]. GitHub. [viewed 11 November 2019]. Available from:
https://github.com/erincatto/Box2D/wiki/FAQ/933830ba42bce329a6697212050da00c383f1e79

Catto, E., 2019b. GitHub - erincatto/Box2D: Box2D is a 2D physics engine for games [online]. GitHub. [viewed 11 November 2019]. Available from: https://github.com/erincatto/Box2D

Christopoulou, E. and Xinogalos, S., 2017. Overview and comparative analysis of game engines for desktop and mobile devices. *International Journal of Serious Games*, *4*(4), p.2017.

Collavizza, H., Michel, C. and Rueher, M., 2016, October. Searching critical values for floating-point programs. In *IFIP International Conference on Testing Software and Systems* (pp. 209-217). Springer, Cham.

Coumans, E., 2019. Bullet Collision Detection & Physics Library: btSequentialImpulseConstraintSolver Class Reference [online]. World Health Organization. [viewed 07 November 2019]. Available from:
https://pybullet.org/Bullet/BulletFull/classbtSequentialImpulseConstraintSolver.html

CPlusPlus, 2019. Tutorials - C++ Tutorials [online]. CPlusPlus. [viewed 13 November 2019]. Available from:
http://www.cplusplus.com/doc/

Cpp Reference, 2019a. cppreference.com [online]. Cpp Reference. [viewed 14 November 2019]. Available from:
https://en.cppreference.com/w/cpp/container/unordered_set

Cpp Reference, 2019b. cppreference.com [online]. Cpp Reference. [viewed 13 November 2019]. Available from:
https://en.cppreference.com/w/

Cpp Reference, 2019c. std::chrono::high_resolution_clock - cppreference.com [online]. CPP Reference. [viewed 13 November 2019]. Available from:
https://en.cppreference.com/w/cpp/chrono/high_resolution_clock

Dawson, B., 2013. Floating-Point Determinism | Random ASCII - tech blog of Bruce Dawson [online]. GitHub. [viewed 12 November 2019]. Available from:
https://randomascii.wordpress.com/2013/07/16/floating-point-determinism/

DevDocs, 2019. DevDocs - C++ documentation [online]. DevDocs. [viewed 11 November 2019]. Available from: https://devdocs.io/cpp/

Devietti, J., Lucia, B., Ceze, L. and Oskin, M., 2009, March. DMP: deterministic shared memory multiprocessing. In ACM SIGARCH Computer Architecture News (Vol. 37, No. 1, pp. 85-96). ACM.

Ducas, L., Galbraith, S., Prest, T. and Yu, Y., 2019. Integral Matrix Gram Root and Lattice Gaussian Sampling without Floats. IACR Cryptology ePrint Archive, 2019, p.320.

Epic Games, 2019. Physics Simulation | Unreal Engine Documentation [online]. Newton Dynamics. [viewed 07 November 2019]. Available from:
https://docs.unrealengine.com/en-US/Engine/Physics/index.html

Erleben, K., 2004. Stable, robust, and versatile multibody dynamics animation. Unpublished Ph. D. Thesis, University of Copenhagen, Copenhagen.

G-Truc Creation, 2019. OpenGL Mathematics [online]. G-Truc Creation. [viewed 12 November 2019]. Available from: https://github.com/nigels-com/glew

Hämäläinen, P., Ma, X., Takatalo, J. and Togelius, J., 2017, October. Predictive physics simulation in game mechanics. In *Proceedings of the Annual Symposium on Computer-Human Interaction in Play* (pp. 497-505). ACM.

Henning, J.L., 2000. SPEC CPU2000: Measuring CPU performance in the new millennium. Computer, 33(7), pp.28-35.

Hillesland, K. and Lastra, A., 2004. GPU floating-point paranoia. *Proceedings of GP2*, *318*.

Hoste, K. and Eeckhout, L., 2008, April. Cole: compiler optimization level exploration. In Proceedings of the 6th annual IEEE/ACM international symposium on Code generation and optimization (pp. 165-174). ACM.

Hummel, J., Wolff, R., Stein, T., Gerndt, A. and Kuhlen, T., 2012, July. An evaluation of open source physics engines for use in virtual reality assembly simulations. In International Symposium on Visual Computing (pp. 346-357). Springer, Berlin, Heidelberg.

HWiNFO, 2019. HWiNFO v6.14 [software]. [Accessed 13 November 2019]. Available from: https://www.hwinfo.com/

Ikits, M., Magallon, M.E., Povalahev, L., Paul. B, 2019. GitHub - nigels-com/glew: The OpenGL Extension Wrangler Library [online]. GitHub. [viewed 12 November 2019].
Available from:
https://github.com/nigels-com/glew

Ivaldi, S., Peters, J., Padois, V. and Nori, F., 2014, November. Tools for simulating humanoid robot dynamics: a survey based on user feedback. In 2014 IEEE-RAS International Conference on Humanoid Robots (pp. 842-849). IEEE.

Jerez, J., 2019. GitHub - MADEAPPS/newton-dynamics: Newton Dynamics is an integrated solution for real time simulation of physics environments. [online]. GitHub. [viewed 07 November 2019]. Available from: https://github.com/MADEAPPS/newton-dynamics

Johnson, J., 2018. Making floating point math highly efficient for AI hardware - Facebook Engineering [online]. Facebook. [viewed 13 November 2019]. Available from:
https://engineering.fb.com/ai-research/floating-point-math/

Juliani, A., Berges, V.P., Vckay, E., Gao, Y., Henry, H., Mattar, M. and Lange, D., 2018. Unity: A general platform for intelligent agents. arXiv preprint arXiv:1809.02627.

Kaeli, D. and Sachs, K., 2009. Computer Performance Evaluation and Benchmarking. Springer.

Kahan, W., 1996. IEEE standard 754 for binary floating-point arithmetic. Lecture Notes on the Status of IEEE, 754(94720-1776), p.11.

Karagoz, P., 2000, March. Adding Constraints to Logic-Based Workflow to Obtain Optimized Schedules. In EDBT PhD Workshop.

Kelly, S.M. and Sigmon, K.A., 2018. The key to key presses: eSports game input streaming and copyright protection. *Interactive Entertainment Law Review*, *1*(1), pp.2-16.

Kneusel, R.T., 2017. Pitfalls of Floating-Point Numbers (and How to Avoid Them). In Numbers and Computers (pp. 117-135). Springer, Cham.
Kreinin, Y., 2008. Consistency: how to defeat the purpose of IEEE floating point [online]. Proper Fixation. [viewed 12 November 2019]. Available from: http://yosefk.com/blog/consistency-how-to-defeat-the-purpose-of-ieee-floating-point.html

Kushnir, V., Koman, B. and Shuvar, R., 2018. Development a Software with Augmented Reality Using Unreal Engine 4.

Learn OpenGL, 2019. LearnOpenGL - Instancing [online]. Learn OpenGL. [viewed 13 November 2019]. Available from: https://learnopengl.com/Advanced-OpenGL/Instancing

León, B., Ulbrich, S., Diankov, R., Puche, G., Przybylski, M., Morales, A., Asfour, T., Moisio, S., Bohg, J., Kuffner, J. and Dillmann, R., 2010, November. Opengrasp: a toolkit for robot grasping simulation. In International Conference on Simulation, Modeling, and Programming for Autonomous Robots (pp. 109-120). Springer, Berlin, Heidelberg.

Lobb, K., Mithra, P. and Berro, M., Nintendo of America Inc, 2004. Real-time replay system for video game. U.S. Patent 6,699,127.

Madougou, S., Varbanescu, A.L., De Laat, C. and Van Nieuwpoort, R., 2016. A tool for bottleneck analysis and performance prediction for gpu-accelerated applications. In 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) (pp. 641652). IEEE.

Martínez-Franco, J.C. and Álvarez-Martínez, D., 2018, December. Physx as a middleware for dynamic simulations in the container loading problem. In *Proceedings of the 2018 Winter Simulation Conference* (pp. 2933-2940). IEEE Press.

Maximilien, E.M. and Williams, L., 2003, May. Assessing test-driven development at IBM. In 25th International Conference on Software Engineering, 2003. Proceedings. (pp. 564-569). IEEE.

McFarlane, J., 2019. GitHub - johnmcfarlane/cnl: A Compositional Numeric Library for C++ [online]. GitHub. [viewed 11 November 2019]. Available from: https://github.com/johnmcfarlane/cnl

Michaelis, K., Meyer, C. and Schwenk, J., 2013, February. Randomly failed! The state of randomness in current Java implementations. In Cryptographers' Track at the RSA Conference (pp. 129-144). Springer, Berlin, Heidelberg.

Microsoft, 2017. /O Options (Optimize Code) | Microsoft Docs [online]. Microsoft. [viewed 12 November 2019]. Available from:

https://docs.microsoft.com/en-us/cpp/build/reference/o-options-optimize-code?view=vs-2019

Microsoft, 2019a. Visual Studio IDE, Code Editor, Azure DevOps, & App Center - Visual Studio [online]. Microsoft. [viewed 12 November 2019]. Available from: https://visualstudio.microsoft.com/

Microsoft, 2019b. C++ programming with Visual Studio Code [online]. Microsoft. [viewed 12 November 2019]. Available from: https://code.visualstudio.com/docs/languages/cpp
Monniaux, D., 2008. The pitfalls of verifying floating-point computations. ACM Transactions on Programming Languages and Systems (TOPLAS), 30(3), p.12.

Newton Dynamics, 2019. Newton Dynamics • About Newton [online]. Newton Dynamics. [viewed 07 November 2019]. Available from: http://newtondynamics.com/forum/newton.php

Nieter, C. and Cary, J.R., 2004. VORPAL: a versatile plasma simulation code. Journal of Computational Physics, 196(2), pp.448-473.

Okano, K., Harauchi, S., Sekizawa, T., Ogata, S. and Nakajima, S., 2019. Consistency Checking between Java Equals and hashCode Methods Using Software Analysis Workbench. IEICE Transactions on Information and Systems, 102(8), pp.1498-1505.

Peitso, L. and Brutzman, D., 2018, June. Defeating lag in network-distributed physics simulations: an architecture supporting declarative network physics representation protocols. In Proceedings of the 23rd International ACM Conference on 3D Web Technology (p. 5). ACM.

Pereira, V., 2014. Learning Unity 2D Game Development by Example. Packt Publishing Ltd.

Perez, A., 2017. Rust and C++ performance on the Algorithmic Lovasz Local Lemma.

Petersen, K., Wohlin, C. and Baca, D., 2009, June. The waterfall model in large-scale development. In International Conference on Product-Focused Software Process Improvement (pp. 386-400). Springer, Berlin, Heidelberg.

Pool, R., 1989. Chaos theory: how big an advance?. Science, 245(4913), p.26.

Rohde, M.M., Crawford, J., Toschlog, M., Iagnemma, K.D., Kewlani, G., Cummins, C.L., Jones, R.A. and Horner, D.A., 2009, April. An interactive physics-based unmanned ground vehicle simulator leveraging open source gaming technology: progress in the development and application of the virtual autonomous navigation environment (VANE) desktop. In Unmanned Systems Technology XI (Vol. 7332, p. 73321C). International Society for Optics and Photonics.

Rönnau, A., Sutter, F., Heppner, G., Oberländer, J. and Dillmann, R., 2013, November. Evaluation of physics engines for robotic simulations with a special focus on the dynamics of walking robots. In 2013 16th International Conference on Advanced Robotics (ICAR) (pp. 17). IEEE.

Sanisalo, J. and Kero P., 2019. GitHub - XMunkki/FixPointCS: A fast, multi-language, multiprecision fixed-point library! [online]. GitHub. [viewed 11 November 2019]. Available from:
https://github.com/XMunkki/FixPointCS

Sawaya, G., Bentley, M., Briggs, I., Gopalakrishnan, G. and Ahn, D.H., 2017, October. FLiT: Cross-platform floating-point result-consistency tester and workload. In 2017 IEEE international symposium on workload characterization (IISWC)(pp. 229-238). IEEE.

Trenkwalder, M., 2019. GitHub - trenki2/FixedPoint: C++ Fixed Point Math Library [online]. GitHub. [viewed 11 November 2019]. Available from: https://github.com/trenki2/FixedPoint

Vasudeva, K. and Bhalla, U.S., 2004. Adaptive stochastic-deterministic chemical kinetic simulations. Bioinformatics, 20(1), pp.78-84.

Veldhuizen, T., 2000. Techniques for scientific C++. *Computer science technical report*, *542*, p.60.

Weaver, V.M., Terpstra, D. and Moore, S., 2013, April. Non-determinism and overcount on modern hardware performance counter implementations. In *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)* (pp. 215-224). IEEE.

Whitehead, N. and Fit-Florea, A., 2011. Precision & performance: Floating point and IEEE 754 compliance for NVIDIA GPUs. *rn (A+ B)*, *21*(1), pp.18749-19424.

Williams, S., Waterman, A. and Patterson, D., 2009. Roofline: An insightful visual performance model for floating-point programs and multicore architectures (No. LBNL2141E). Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States).

Wolfram Research, Inc., 2019. LearnOpenGL - Instancing [online]. Wolfram Research, Inc. [viewed 13 November 2019]. Available from: http://mathworld.wolfram.com/Mean.html

ZINT, Michaela., 2019. *Power Analysis, Statistical Significance, & Effect Size* [online]. meera.
[viewed 15 November 2019]. Available from:
http://meera.snre.umich.edu/power-analysis-statistical-significance-effect-size